28 02 2024
Wednesday

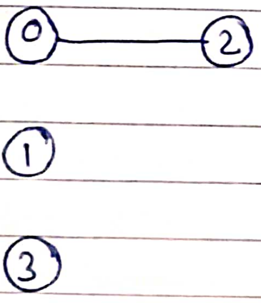
* **Number of Provinces (leetcode - 547)**

There are n cities. If city a is directly connected with b and b with c, then a is indirectly connected with c.
A province is a group of directly or indirectly connected cities.
Given a matrix isConnected. If isConnected[i][j] = 1, it means connected, if 0, then not connected.
Return total provinces.

I/P - isConnected =

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 \end{array}$$

Graph for this matrix will be



O/P - 3
Clearly, there are 3 provinces in total.

Explanation - We just have to find the number of disconnected components in adjacency matrix.

Code -

```
class Solution{
public:

void dfs (int srcNode, vector<vector<int>>&isConnected,
            unorderd_map<int, bool>& visited)
{
      visited [srcNode]=true;

      int row = srcNode;
      int col = isConnected[0].size;

      for(int nbr=0; nbr<col; nbr++){
          if (isConnected[row][nbr]==1 && !visited[nbr]){
                dfs (nbr, isConnected, visited);
          }
      }
  }

int findCircleNum (vector<vector<int>>& isConnected)
{
     unordered_map<int, bool> visited;
     int noOfProvinces = 0;

     for(int node=0; node<isConnected.size(); node++){
          if(!visited[node]){
               noOfProvinces ++;
               dfs (node, isConnected, visited);
          }
     }
     return noOfProvinces;
  }
};
```

* **Number of islands (leetcode-200)**

Given m×n 2D binary grid represents a map of 1's (land) and 0's (water), return no. of islands.
An island is surrounded by water & formed by connecting adjacent lands horizontally or vertically.

I/P- grid = [["1", "1", "0", "0", "0"],
["1", "1", "0", "0", "0"],
["0", "0", "1", "0", "0"],
["0", "0", "0", "1", "1"]]

O/P- 3

xplanation- The "1" here represents the piece of land.
If a particular piece of land is surrounded by land from other 4 directions, then we will consider it as one islands.
Here too, we have to find the number of disconnected components.

for that, traverse the whole grid and if a piece of land is not visited, then visit it. We can visit the grid either by BFS or DFS.
Here, we are doing it with BFS.
So, whenever the queue gets empty, we will increment the count of islands by 1.

Code -

```
class Solution{
public:

bool isSafe(int newX, int newY, vector<vector<char>>
            & grid, map<pair<int,int>, bool>& visited)
{
  if(newX>=0 && newY>=0 && newX <grid.size() &&
     newY< grid[0].size() && grid[newX][newY]=='1'
     && !visited[{newX,newY}]) return 1;

  else   return false;
}

//bfs traversal
void  bfs(int srcX, int srcY,vector<vector<char>>&grid,
          map<pair<int, int>, bool>&visited)
{
   //make queue
   queue<pair<int,int>>q;

   // maintain initial state
   q.push({srcX,srcY});
   visited[{srcX,srcY}]= true;

   while (!q.empty()){
     auto frontNode= q.front();
     q.pop();

     int currX= frontNode. first;
     int currY= frontNode. second;
// to move left, right, up, down in grid
 vector<int> dx = {0, 0, -1, 1};
 vector<int> dy = {-1, 1, 0, 0};

 for (int i=0; i<4; i++){
        int newX = currX + dx[i];
        int newY = currY + dy[i];
        if (isSafe(newX, newY, grid, visited)){
            q.push({newX, newY});
            visited[{newX, newY}] = true;
        }
    }
  }
}

int  numIslands (vector<vector<char>>& grid){
      map<pair<int, int>, bool> visited;
      int noOfIslands = 0;

      for (int i=0; i<grid.size(); i++){
        for(int j=0; j<grid[0].size(); j++){
           if (!visited[{i,j}] && grid[i][j] == '1'){
               bfs(i, j, grid, visited);
               noOfIslands ++;
           }
        }
      }
      return noOfIslands;
   }
};
```